Riklam (Plataforma de Descubrimiento Local Inteligente)

Miranda Alfaro Luis Eduardo, Briseño Chong Jesús, Castillo Rocha Bryan Alejandro

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS, (CUCEI, UDG)

luis.miranda6666@alumnos.udg.mx

jesus.briseno5273@alumnos.udg.mx

bryan.castillo4566@alumnos.udg.mx

***Abstract*— Antecedentes: Las plataformas actuales de descubrimiento local generan "fatiga de decisión". El objetivo es diseñar una plataforma inteligente que resuelva esta desconexión mediante el descubrimiento proactivo de "joyas ocultas". Métodos: Se diseñó una arquitectura de microservicios en la nube con frontend móvil y web (React/React Native). El núcleo es un sistema de recomendación híbrido y sensible al contexto (CARS) usando Python, Node.js, MongoDB y PostgreSQL+PostGIS, validado con un prototipo en Figma. Resultado Esperado del Proyecto: Se espera validar la viabilidad de la arquitectura. La simulación proyecta que los flujos críticos son viables, con una tasa de interacción >40% en "joyas ocultas" y alta usabilidad. Un hallazgo clave es la necesidad de pre-cálculo para cumplir el rendimiento. Conclusiones: El diseño integrado es técnicamente factible y viable. Este proyecto documenta una base sólida para la futura implementación del sistema.**

Palabras claves – Proyecto modular, descubrimiento local, inteligencia artificial, sistemas de recomendación, arquitectura de microservicios, joyas ocultas.

Repositorio de código: https://github.com/Brycasti/Proyecto-Modular

Versión actual del código: v1.0.0

I. Introducción

Las plataformas de descubrimiento actuales, como Google Maps o Yelp, presentan un problema de "fatiga de decisión". Muestran *todo* lo disponible, saturando al usuario con cientos de opciones genéricas y turísticas. Esto provoca que el residente local, que busca una experiencia auténtica y alineada a sus gustos, termine frecuentando los mismos lugares de siempre.

Este proyecto, la " Riklam (Plataforma de Descubrimiento Local Inteligente)", aborda este problema desde un enfoque "anti-turístico". La solución no es un directorio, sino un curador personal de experiencias que se centra en el descubrimiento proactivo de "joyas ocultas" mediante un motor de IA que generará un feed diario y personalizado, similar al "Efecto Spotify". El sistema aprenderá del usuario y su contexto (hora, día, ubicación) para mostrar solo lo que amará, reconectando al residente con su ciudad y apoyando a la economía local.

II. Trabajos Relacionados

El diseño de este proyecto se sitúa en la intersección de varias disciplinas clave en la informática: los Sistemas de Recomendación (RS), los Sistemas de Información Geográfica (GIS) y el manejo del "problema del arranque en frío".

A. Sistemas de Recomendación (RS)

Los sistemas de recomendación tradicionales se dividen en dos categorías principales:

1. Filtrado Basado en Contenido (Content-Based): Sugiere ítems basándose en las propiedades de esos ítems y las preferencias pasadas de un usuario, si a un usuario le gustaron restaurantes de "comida italiana", el sistema recomendará más "comida italiana".

2. Filtrado Colaborativo (Collaborative Filtering): Sugiere ítems basándose en las preferencias de usuarios "similares", si al Usuario A y al Usuario B les gustaron 10 restaurantes en común, el sistema recomendará al Usuario A el restaurante número 11 que al Usuario B le encantó.

Ambos enfoques presentan debilidades, el basado en contenido sufre de sobre-especialización (nunca descubre nada nuevo), mientras que el colaborativo sufre del "problema del arranque en frío" (Cold Start): no puede hacer recomendaciones a un usuario nuevo o sobre un negocio nuevo, ya que no existen interacciones previas.

B. El Problema del "Cold Start" y los RS Híbridos

El "problema del arranque en frío" es el desafío más grande para una plataforma nueva, nuestro proyecto está diseñado específicamente para mitigar esto mediante un enfoque híbrido:

- Se combina el *filtrado basado en contenido* (obtenido del perfil de usuario y sus preferencias explícitas) para nuevos usuarios.

- A medida que el usuario interactúa, se introduce el *filtrado colaborativo* para mejorar la precisión y descubrir "joyas" (el "Efecto Spotify").

C. Sistemas Sensibles al Contexto (CARS) y Recomendación de Puntos de Interés (POI)

Un sistema de recomendación de lugares no puede funcionar igual que uno de películas (como Netflix). La recomendación de un "Punto de Interés" (POI) depende fundamentalmente del contexto, como señalan Adomavicius y Tuzhilin, un Sistema de Recomendación Sensible al Contexto (Context-Aware Recommender System, o CARS) es necesario.

El contexto es multidimensional: no es solo *quién* (usuario) y *qué* (lugar), sino también *cuándo* (hora del día, temporada), *dónde* (ubicación actual del usuario) y *con quién* (solo, pareja, amigos). La elección de PostgreSQL con PostGIS es una decisión de arquitectura directa para soportar eficientemente este tipo de consultas geoespaciales y de proximidad, que son críticas para un CARS de POI.

D. Aplicaciones Comerciales y la Brecha del Mercado

Las plataformas existentes como *Google Maps* y *TripAdvisor* son excelentes guías turísticas y GIS, pero sus motores de recomendación son rudimentarios y se basan en la popularidad general. *Yelp* y *Foursquare* se acercan más al descubrimiento social, pero sus modelos de negocio (basados en publicidad pagada) comprometen la imparcialidad, creando un "sesgo de popularidad" que entierra a las "joyas ocultas".

Nuestro proyecto se posiciona para llenar esta brecha, utilizando un motor CARS híbrido cuyo objetivo algorítmico no es mostrar lo más popular, sino lo más afín y novedoso.

III. Descripción del Desarrollo del Proyecto Modular

A. Metodología de Trabajo

El proyecto se gestionó siguiendo un enfoque ágil, con un equipo de tres integrantes (Bryan Castillo, Eduardo Miranda, Jesús Briseño) asumiendo roles definidos (Diseño/Frontend, IA/Base de Datos, y Backend/DevOps). Se planificaron revisiones semanales ("Sprints") para monitorear el avance de los entregables de la Estructura de Desglose del Trabajo (WBS 1.1.3). La gestión de tareas se centralizó en herramientas como Trello/Jira, y todo el control de versiones se diseñó para ser manejado a través de Git.

B. Requerimientos Principales

El diseño se centró en tres requerimientos críticos identificados en la fase de análisis de factibilidad, los cuales son indispensables para el éxito del proyecto:

RF-5 (Feed de Descubrimiento Proactivo): El núcleo de la propuesta de valor y el principal diferenciador del sistema.

RF-9 (Gestión de Perfil de Negocio): El portal autogestionable, crítico para escalar la base de datos de contenido.

RNF-1 (Usabilidad): Esencial para la adopción tanto de usuarios (que deben proveer sus preferencias) como de negocios (que deben poder publicar fácilmente).

C. Tecnologías Utilizadas (Diseño)

Para cumplir con los requerimientos técnicos, se seleccionó un stack de tecnologías de código abierto, modernas y escalables (documentadas en la "Selección de componentes y materiales"):

Frontend: React Native (o Flutter) para la App Móvil (iOS/Android) y React (o Vue.js) para el Portal Web.

Backend: Microservicios desarrollados en Node.js (con Express.js).

Motor IA: Un microservicio independiente en Python, utilizando librerías como TensorFlow o PyTorch.

Bases de Datos: Un enfoque dual con MongoDB (para perfiles flexibles y preferencias) y PostgreSQL con PostGIS (para datos geoespaciales y búsquedas de proximidad).

Infraestructura: Proveedores Cloud como AWS, Google Cloud o Azure, utilizando sus niveles gratuitos (*Free Tier*) para el despliegue inicial.

D. Diseño de la Arquitectura

La arquitectura del sistema fue diseñada para ser desacoplada y escalable.

1) Capa de Presentación (Frontend)

Desarrollada para dos tipos de usuario: la *App Móvil* (en React Native o Flutter) para el usuario final (consumidor), enfocada en el descubrimiento; y el *Portal Web* (en React o Vue.js) para los negocios, permitiendo la autogestión de su perfil y publicaciones.

2) Capa de Lógica de Negocio (Backend)

Expuesta como una API RESTful, está compuesta por microservicios desacoplados, incluyendo:

Microservicio de Usuarios: Maneja autenticación, perfiles y preferencias.

Microservicio de Negocios: Gestiona la información, publicaciones y perfiles de negocios.

Microservicio de Búsqueda: Provee filtros y geolocalización (usando PostGIS).

Motor de Recomendación (IA): Un servicio independiente que genera el feed proactivo.

3) Capa de Datos

Se optó por un enfoque dual: *MongoDB* (NoSQL) para almacenar perfiles flexibles, preferencias y contenido no estructurado; y *PostgreSQL + PostGIS* (Relacional/Geoespacial) para gestionar ubicaciones precisas y consultas de proximidad.

4) Motor de Recomendación IA

Es el núcleo del proyecto. Utiliza una fórmula de "Discovery Score" que balancea múltiples factores para priorizar joyas ocultas:

Score = (Match x 0.4) + (Prox. x 0.25) + (Gem x 0.2) + (Ctx. x 0.15)

Donde *Gem Factor* se define como:

Gem = Calidad x (1 - Índice de Popularidad)

E. Pruebas Realizadas (Simulación)

Dado que este es un proyecto de documentación y diseño, la validación se realizó mediante modelado y simulación. Se utilizó un prototipo interactivo de alta fidelidad en Figma para simular los flujos de usuario y validar la viabilidad de los conceptos.

Fig. 1 Selección de preferencias

Fig. 2 Finalización de perfil

Fig. 3 Ubicaciones cerca de tu zona preferida

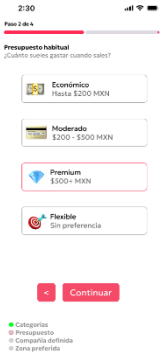

Fig. 4 Presupuestos adaptados para cada persona

El plan de implementación (WBS 1.6) también define las pruebas futuras necesarias para la implementación, incluyendo Pruebas Unitarias (Backend/Frontend), Pruebas de Integración (APIs) y Pruebas de Usabilidad en dispositivos reales.

F. Proceso de Implementación (Diseño)

El "cómo se hizo" este diseño siguió un proceso lógico documentado en las entregas del curso:

Definición: Se identificó el problema (fatiga de decisión) y los objetivos del proyecto.

Factibilidad: Se analizó la viabilidad técnica (RT), económica (costo nulo) y de recursos (equipo de 3).

Requerimientos: Se definieron los RF, RNF y RT (documento de requerimientos).

Arquitectura: Se diseñó la arquitectura preliminar (hardware y software).

Fig. 5 Arquitectura de hardw

Fig. 6 Arquitectura de software

Simulación: Se modeló el sistema y se validó el diseño con prototipos (Figma).

Documentación: Se consolidó toda la información en este documento.

G. Repositorio de Código

Aunque el proyecto se centra en la documentación, el repositorio público para la futura implementación del código se encuentra en: https://github.com/Brycasti/Proyecto-Modular

IV. Resultados Esperados del Proyecto

Para la validación de la arquitectura y el diseño conceptual, se define un plan de modelado y simulación (basado en el prototipo de Figma) para probar el sistema contra las variables clave (perfil de usuario, contexto y parámetros del motor IA).

Se espera que la simulación y el análisis de viabilidad validen los flujos críticos. Se proyecta que el flujo de descubrimiento proactivo logre el "Efecto Spotify" deseado, estableciendo una métrica objetivo de tasa de interacción superior al 40% en "joyas ocultas" (comparado con recomendaciones estándar). Se anticipa que el portal de negocios alcance una alta tasa de completitud de perfiles (objetivo > 85%) en la primera sesión, confirmando su alta usabilidad.

Un hallazgo técnico clave identificado durante la fase de diseño es la necesidad de implementar el pre-cálculo de recomendaciones para garantizar el cumplimiento del requisito de rendimiento de tiempos de carga menores a 3 segundos, un desafío común en sistemas CARS.

V. Conclusiones y Trabajo a Futuro

El diseño preliminar integrado (arquitectura, simulación y componentes) documentado en este proyecto es coherente, consistente y viable, la contribución clave es la validación de una arquitectura escalable y técnicamente factible capaz de soportar el concepto central. El análisis de la simulación ha mitigado el riesgo de usabilidad y ha confirmado que el diseño propuesto puede, en efecto, resolver la fatiga de decisión y el sesgo de popularidad identificados en la introducción.

El trabajo a futuro se centrará en la implementación de este diseño y las recomendaciones principales incluyen:

Optimización del Motor IA: Implementar la técnica de pre-cálculo de *feeds* (identificada en la Sección IV) para asegurar el rendimiento.

Mitigación del "Arranque en Frío": Explorar métodos alternos para la baja densidad de negocios, como la integración de datos de APIs de terceros mientras se puebla la base de datos propia.

Evolución del Algoritmo: Iterar sobre el motor de recomendación para incluir más factores contextuales (como el clima) que no se incluyeron en el diseño inicial.

Referencias

[1] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," Computer, vol. 42, no. 8, pp. 30–37, Aug. 2009.

[2] F. Ricci, L. Rokach, and B. Shapira, Eds., Recommender Systems Handbook. Springer, 2011.

[3] S. Newman, Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 2015.

[4] M. Ye, P. Yin, and W.C. Lee, "Location-based and preference-aware recommendation," in Proc. 20th ACM SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems, 2012, pp. 109–118.

[5] (2025) React Native Documentation. [Online]. Available: https://reactnative.dev/

[6] (2025) The MongoDB Manual. [Online]. Available: https://www.mongodb.com/docs/manual/

[7] (2024) PostGIS Development Group. PostGIS 3.4.0 Manual. [Online]. Available: https://postgis.net/docs/

[8] G. Adomavicius and A. Tuzhilin, "Context-aware recommender systems," in Recommender systems handbook. Springer, 2011, pp. 217–253.

[9] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, "Collaborative filtering recommender systems," in The adaptive web. Springer, 2007, pp. 291–324.

[10] (2025) Node.js Foundation. Node.js Documentation. [Online]. Available: https://nodejs.org/en/docs/

[11] (2025) Meta Platforms, Inc. React Documentation. [Online]. Available: https://react.dev/

[12] (2025) Google. TensorFlow Documentation. [Online]. Available: https://www.tensorflow.org/api_d